Tips!
Serge Iovleff

1 Tips: Centering an array

It’s very easy to center a data set using high level templated expressions and statistical functors.

Listing 1: Example

```cpp
#include "STKpp.h"
using namespace STK;

int main(int argc, char *argv[])
{
    CArrayXX A(100, 5);
    Law::Normal law(1, 2);
    A.rand(law);
    // call column statistical functions
    stk_cout << _T("min(A) =") << Stat::min(A);
    stk_cout << _T("max(A) =") << Stat::max(A);
    stk_cout << _T("mean(A) =") << Stat::mean(A);
    // center the array
    stk_cout << _T("Centering...
" "
    A -= Const::VectorX(100) * Stat::mean(A);
    // call column statistical functions with all the columns of A
    // centered
    stk_cout << _T("min(A) =") << Stat::min(A);
    stk_cout << _T("max(A) =") << Stat::max(A);
    stk_cout << _T("mean(A) =") << Stat::mean(A);
    }
```

Listing 1: Output

```
min(A) = -4.66112 -3.1092 -2.76891 -4.65294 -5.47609
max(A) = 4.17889 5.89107 5.08115 6.39462 5.17813
max(A.abs()) = 4.66112 5.89107 5.08115 6.39462 5.47609
mean(A) = 0.963606 1.26585 1.26198 1.05056 0.723175
Centering...
min(A) = -5.62472 -4.37505 -4.0309 -5.7035 -6.19926
max(A) = 3.21529 4.62522 3.81916 5.34406 4.46495
max(A.abs()) = 5.62472 4.62522 3.81916 5.34406 4.46495
mean(A) = 2.57572*10^-16 -3.93019*10^-16 2.13163e-16 -3.4639e-16 -1.15463e-16
```

the expression

```
Const::Vector<Real>(100) * mean(A)
```

represents the matrix multiplication of a column vector of 1 with 100 rows and of row vector with the mean of each column of A.

Note:
For each column of the array A we can get the maximal value in absolute value using \texttt{max(A.abs())}. It is possible to use functors mixed with unary or binary operators.

2 Tips: Compute the mean for each column of an array

You can easily get the mean of a whole vector or a matrix containing missing values using the expression

```
CArray<Real> A(100, 20);
Law::Normal law(1, 2);
A.rand(law);
Real m = A.meanSafe();
```

In some cases you may want to get the mean for each column of an array with missing values. You can get it in a `PointX` vector using either the code

```
PointX m;
PointX::meanByCol(A.safe());  // mean(A.safe()); is shorter
```

or the code

```
Array2DPoint<Real> m;
Array1DPoint::move(Stat::mean(A.safe()));
```
The method `A.safe()` will replace any missing (or NaN) values by zero. In some cases it’s not sufficient, Suppose you know your data are all positive and you want to compute the log-mean of your data. In this case, you will rather use

```cpp
m = Stat::mean(A.safe(1.).log());
```

and all missing (or NaN) values will be replaced by one.

**Note:**
You can also compute the variance. If you want to compute the mean of each row, you will have to use the functor `Stat::meanByRow`. In this latter case, you get a `VectorX` as result.

### 3 Tips: Compute the mean and the variance of multidimensionnal data

You can easily compute the mean and the variance matrix of multidimensional data. Assume we are handling this kind of data

```cpp
// values (b,g,r,ir)
typedef CArrayVector<double, 4> Spectrum;
```

repeated in space and time. The data are stored in an array

```cpp
// array of values
typedef CArray<Spectrum> ArraySpectrum;
ArraySpectrum datait;
```

and we want to compute at each time the (multidimensional) mean of this data set. This can be used using the following code :

```cpp
// array of mean values
typedef CArrayPoint<Spectrum> PointSpectrum;
PointSpectrum mut(datait.cols());
for (int t = datait.beginCols(); t < datait.endCols(); ++t)
{
    mut[t] = 0.;
    for (int i = datait_.beginRows(); i < datait_.endRows(); ++i)
    {
        mut[t] += datait_(i,t);
    }
    mut[t] /= datait_.sizeRows();
}
```

The variance matrix (using numerical correction) can be computed using the following code :

```cpp
// covariances values (b,g,r,ir)
typedef CArraySquare<double, 4> CovSpectrum;
// array of mean values
typedef CArrayPoint<CovSpectrum> PointCov;
PointSpectrum sigmat(datait.cols());
for (int t = datait.beginCols(); t < datait.endCols(); ++t)
{
    CovSpectrum var; var=0.0;
    Spectrum sum = 0.0;
    for (int i = datait_.beginRows(); i < datait_.endRows(); ++i)
    {
        Spectrum dev;
        sum += (dev = datait(i,t) - mut[t]);
        var += dev*dev.transpose();
    }
    sigmat[t] = (var - ((sum*sum.transpose())/datait_.sizeCols()))/datait_.sizeCols();
}
```

STK++ handles transparently the multidimensional nature of the data.